

Procedure *INIT-PATH*  
*{Invoked when the node comes up.}*

1. Initialize all tables.

2. Run *PATH* algorithm.

End *INIT-PATH*

Algorithm *PATH*

*{Invoked when a message M is received from neighbor k, or an adjacent link to k has changed or when a node is initialized.}*

1. Run *NTU* to update neighbor tables.

2. Run *MTU* to update main tables.

3. For each destination j marked as *changed*,  
 Add update entry  $[j, D_j^i, p_j^i]$  to the new message *M'*.

4. Within finite amount of time, send message *M'* to  
 each neighbor.

End *PATH*

Procedure *NTU*  
*{Called by PATH to process an event.}*

1. If event is a message M from neighbor  $k, \tilde{p} = p_j^k$ 
  - a. For each entry  $[j, d, p]$  in M //Note  $d = D_j^k, \tilde{p} = p_j^k$   
 Set  $D_{jk}^i \leftarrow d$  and  $p_{jk}^i \leftarrow p$
  - b. For each destination  $j$  with an entry in M,  
 Remove existing links  $(n, j)$  in  $T_k^i$  and add new  
 link  $(m, j, d)$  to  $T_k^i$ , where  $d = D_{jk}^i - D_{mk}^i$   
 and  $m = p_{jk}^i$
2. If the event is an adjacent link-status change, update  $l_k^i$  and  
 clear neighbor tables of  $k$ , if link is down.

End *NTU*

**FIG. 2**

**FIG. 1**

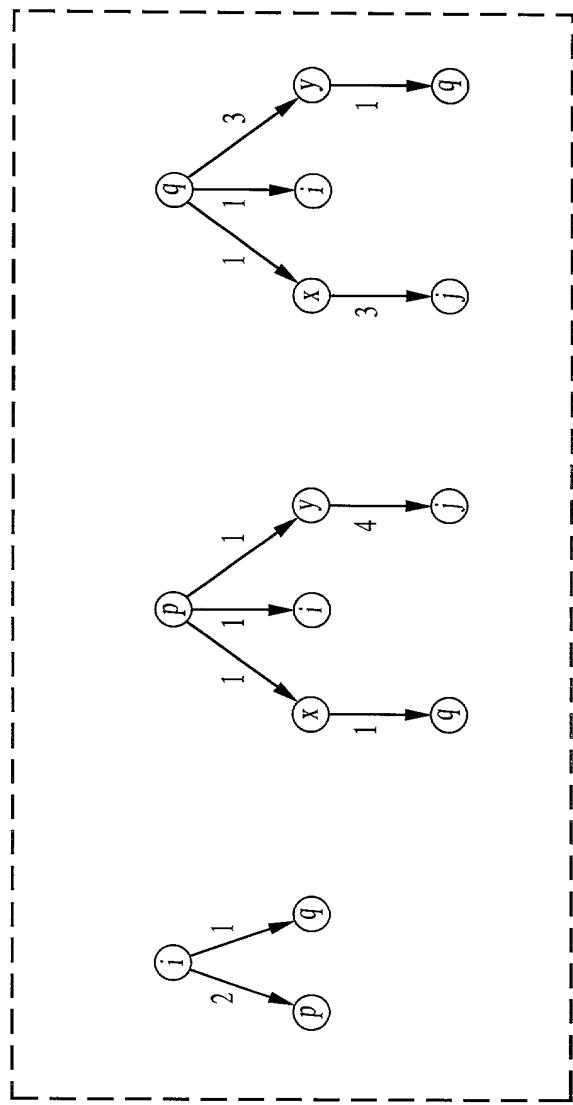
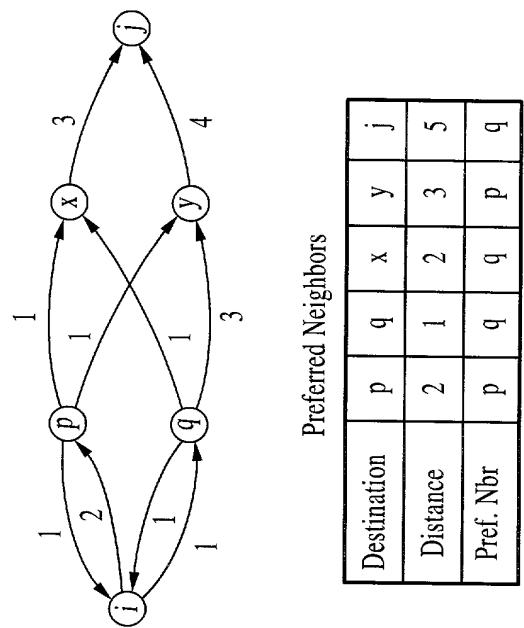
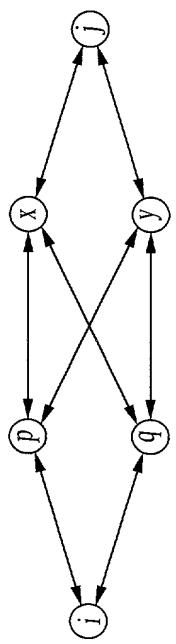
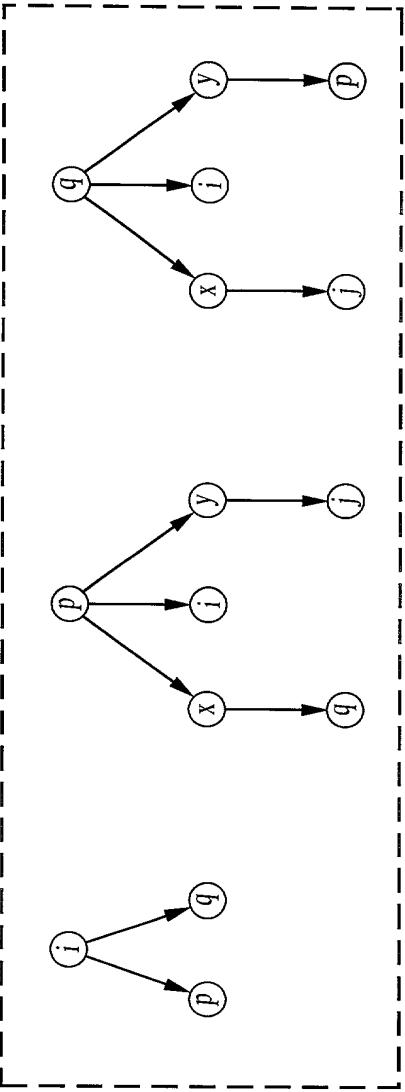
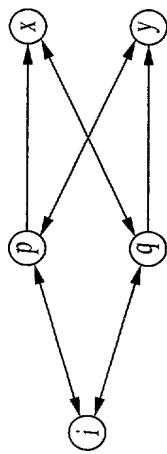


FIG. 3A

**FIG. 3B**

**FIG. 4A****FIG. 4B****FIG. 4C**

```

Procedure MTU
  1. Clear link table  $T'$ 
  2. For each node  $j \neq i$  occurring in at least one  $T'_k$ 
    a. Find  $MIN \leftarrow \min\{D'_k + l'_k | k \in N'\}$ 
    b. Let  $n$  be such that  $MIN = (D'_{jn} + l'_{jn})$ . Ties are
       broken consistently. Neighbor  $n$  is the preferred neighbor
       for destination  $j$ . For each link  $(j, v, d)$  in  $T'_n$ 
       Add link  $(j, v, d)$  to  $T'$ 
  3. Update  $T$  with each link  $l'_k$ 
  4. Run Dijkstra's shortest path algorithm on  $T'$  to
     find new  $D'_j$ , and  $p'_j$ 
  5. For each destination  $j$ , if  $D'_j$  or  $p'_j$  changed from
     previous value, set changed and report-it flags for  $j$ .
End MTU

```

**FIG. 5**

Procedure *INIT-MPATH*  
*{Invoked when the node comes up.}*  
 1. Initialize tables and run MPATH.  
 End *INIT-MPATH*

Algorithm M P A T H  
*{Invoked when a message M is received from neighbor k,  
 or an adjacent link to k has changed.}*  
 1. Run NTU to update neighbor tables.  
 2. Run MTU to obtain new  $D_j^i$  and  $p_j^i$ .  
 3. If node is *PASSIVE* or node is *ACTIVE*  $\wedge$  last reply arrived,  
 Reset *goactive* flag.  
 For each destination  $j$  marked as *report-it*,  
 a.  $FD_j^i \leftarrow \min\{D_j^i, RD_j^i\}$   
 b. If  $D_j^i > RD_j^i$ , Set *goactive* flag.  
 c.  $RD_j^i \leftarrow D_j^i$   
 d. Add  $[j, RD_j^i, p_j^i]$  to message  $M'$ .  
 e. Clear *report-it* flag for  $j$ .  
 Otherwise, the node is *ACTIVE* and waiting for more replies,  
 For each destination  $j$  marked as *changed*,  
 f.  $FD_j^i \leftarrow \min\{D_j^i, FD_j^i\}$   
 4. For each destination  $j$  marked as *changed*,  
 a. Clear *changed* flag for  $j$   
 b.  $S_j^i \leftarrow \{k | D_{jk}^i < FD_j^i\}$   
 5. For each neighbor  $k$ ,  
 a.  $M'' \leftarrow M'$   
 b. If event is *query* from  $k$ , Set *reply* flag in  $M''$ .  
 c. If *goactive* set, Set *query* flag in  $M''$ .  
 d. If  $M''$  non-empty, send  $M''$  to  $k$ .  
 6. If *goactive* set, become *ACTIVE*, otherwise  
 become *PASSIVE*.  
 END *MPATH*

## FIG. 6